It is not hard to learn more. What is hard is to unlearn when you discover yourself wrong. ~ *Martin H. Fischer*

CS1010 Programming Methodology
Week 7: One-dimensional Arrays

To students:
Three weeks of discussion sessions have passed and so we have removed the usual preamble, since you should already know what is expected of you by now.

I. Discussion on Lab #2
Your DL will discuss lab #2 exercises with you. Please get your doubts cleared.

II. Arrays basics

The questions in this section are meant for you to attempt on your own before your discussion session. You **must** do these questions on your own. **They will not likely be discussed in class**, as these are the basics which we expect you to know. You can verify the answers yourself, but if you do have doubts, please post them on the IVLE forum.

1. **Spot the errors.**

   (a) Spot the errors in **q1a.c**. Are there any compilation errors?

```
#include <stdio.h>

int main(void) {
   float[5] values;
   int i;

   for (i=1; i<=5; i++)
      values[i] = 2.5 * i;

   for (i=1; i<=5; i++)
      printf("%f ", values[i]);
   printf("\n");

   return 0;
}
```

   Can you print all the elements in an array by replacing the second **for** loop in the program above with just the following statement?

```
printf("%f\n", values); // to print ALL elements
```

   (b) Spot the errors in **q1b.c**. Are there any compilation errors?

```
#include <stdio.h>

float sumArray(float, int);

int main(void) {
   float prices[6], total;
   prices = { 10.2, 5.3, 4.4, 6.8, 7.7, 9.5 };

   sumArray(prices[6], 6);
   printf("Total = %f\n", total);

   return 0;
}

// Compute the sum of elements in array arr
// size: number of elements in arr
// Precond: size > 0
float sumArray(float arr, int size) {
   int i;
   float sum = 0.0;

   for (i=0; i<size; i++)
      sum += prices[i];

   return sum;
}
```

Assuming that the program is corrected. In the main() function, what if we (i) pass 3 to the **size** parameter of sumArray() function; (ii) pass 10 to the **size** parameter of sumArray() function? Will there be compilation error?

2. **Manual tracing.**
Trace the programs manually and write out their outputs. Then run the programs to verify whether your outputs are correct.

The function **printArray()** used in **q2a.c** and **q2b.c** is shown below:

```
// Print array arr with size elements
// Precond: size > 0
void printArray(int arr[], int size) {
   int i;

   for (i=0; i<size; i++)
      printf("%d ", arr[i]);
   printf("\n");
}
```

(a) Program **q2a.c**

```
#include <stdio.h>
#define LENGTH 10

void printArray(int [], int);
void arrange(int [], int);

int main(void) {
   int numbers[] = {1,2,3,4,5,6,7,8,9,10};

   arrange(numbers, LENGTH);
   printArray(numbers, LENGTH);

   return 0;
}

void arrange(int arr[], int size) {
   int i;

   for (i=2; i<size; i++) {
      arr[i-2] += arr[i];
      arr[i-1] += arr[i];
   }
}

// printArray() omitted here for brevity
```

You want to call **printArray()** somewhere in the program for checking/debugging purpose. Where is a good place to put it and how to call it?

(b) Program **q2b.c**

```
#include <stdio.h>
#define LENGTH 10

void printArray(int [], int);
void process(int [], int);

int main(void) {
   int numbers[] = {5,1,3,9,7,8,2,0,6,4};

   process(numbers, LENGTH);
   printArray(numbers, LENGTH);

   return 0;
}

void process(int arr[], int size) {
   int i;

   for (i=0; i<size; i++) {
      arr[i] = arr[arr[i]];
   }
}

// printArray() omitted here for brevity
```

You want to call **printArray()** somewhere in the program for checking/debugging purpose. Where is a good place to put it and how to call it?

III. Arrays

Important notes:

As we are basing on ANSI C (or C90), we do NOT allow variable-length arrays. Hence the following code is not permitted, because at compilation time the system wouldn't know what value **n** contains and hence the size of the array is unknown:

```
int n, a[n];
printf("Enter n: ");
scanf("%d", &n);
. . .
```

However, as gcc is C99-compliant (C99 is a newer standard for C, which permits variable-length arrays), the code above can be compiled without warning, even if –Wall option is used.

You may compile with the –pedantic option instead, in which case a warning message will be issued.

For all problems on arrays, <u>we will specify the maximum size of an array</u>, so that you can declare the array with the right size. We will not accept the use of variable-length arrays.

3. This is a true story. On 25 August 2010 we received an email from a lecturer about the following program **q3.c**, which when run, gives an infinite loop! Give a possible explanation. What is the moral of the story?

```
#include <stdio.h>
int main(void) {
   double arr[] = { 1.1, 2.2, 3.3, 4.4 };
   int i;

   for (i=0; i<=4; i++) {
      printf("%d\n", i);
      arr[i] = 0;
   }

   return 0;
}
```

4. **Logical thinking.**
   An array is a collect of data. It is very common to ask the following two questions about a collection: (a) do all the data in the collection share a certain property? (b) does there exist one datum that has a certain property? The former is a *universal* question, and the later an *existential* question.

   For example, "are all the values in the array non-negative?" is a universal question; "is there one value in the array that is negative?" is an existential question. In this case, the two questions are actually the same, hence we can transform one into another.

   Write a function **nonNegative(int arr[], int size)** that returns 1 (true) if all the elements in arr[0]… arr[size-1] are non-negative; or returns 0 (false) otherwise. You may assume that the array has at least one element.

5. **Logical thinking.**
   Given an array of integers, write a function **isSorted(int arr[], size)** that returns 1 (true) if the array **arr** is sorted in non-decreasing order, or returns 0 (false) otherwise. You may assume that the array has at least one element.

   For example, 3, 12, 15, 18 and -5, 8, 8, 10 are in non-decreasing order, but 4, 6, 9, 7, 12 is not.

   Do you see any similarity between this question and Q4? (Yes, Computational Thinking: **Pattern Recognition**!) For this question, what is the property you have "abstracted" out to check?

6. **Checking duplicates.**
   Write a program **duplicates.c** to fill an integer array with *n* ($1 \le n \le 1000$) random integers whose values are in the range [*lower*, *upper*] where *n*, *lower* and *upper* are inputs from user. Moreover, 0 < *lower* < *upper*. Your program then computes the total number of duplicates in the array.

   For example, if a 15-element array contains the values { 97, 12, 45, 97, 23, 12, 53, 97, 30, 30, 10, 53, 8, 1, 53 }, then there are altogether 8 duplicates (three 97s, two 12s, and three 53s).

   (We will revisit this question after we have covered sorting. For the moment, do not use sorting in your solution.)